READER
FACULTEIT TECHNISCHE NATUURWETENSCHAPPEN

MATLAB INSTRUCTIE CURCUS 2009 (AT-TN-CT)

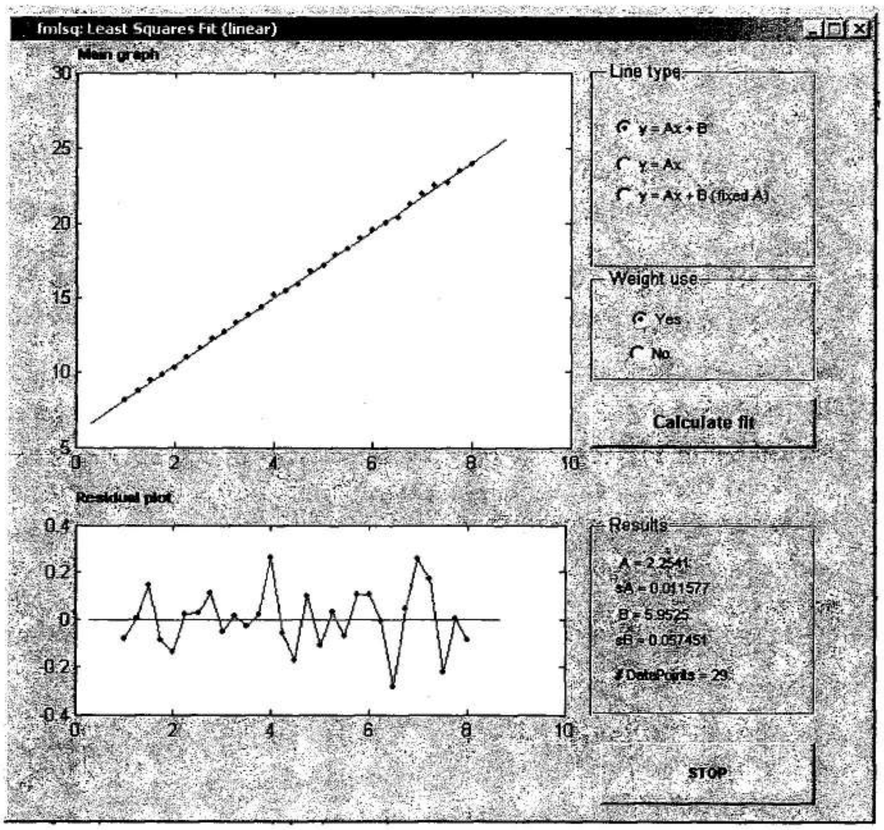
AUGUSTUS 2009
VAKCODE: 390140/140580/139911
OPLAGE: 170
PRIJS: € 5,00

765

NIVERSITEIT TWENTE.

MATLAB INSTRUCTIE
cursus 2009

(AT-TN-CT)




```
Array vullen (als rij)

x = [1   2   2.5   3   4]          % elementen afzonderlijk
y = 1 : 0.5 : 3                    % stapgrootte = 0.5
z = linspace(0,10,6)               % aantal punten = 6
```

Inhoudsopgave

M0 Inleiding 3

M1 Eenvoudige berekeningen 4

1.1. Directe berekeningen uitvoeren 4
1.2. Rekenen met variabelen 4
1.3. Toelichtend commentaar (%) 5
1.4. Echo onderdrukken (;) 5
1.5. Wiskundige functies 5
1.6. Matlab Wokspace 6
1.7. M-files 7

M2 Werken met arrays 8

2.1. Arrays invoeren 8
2.2. Rekenen met arrays (eenvoudig) 9
2.3. Rekenen met arrays (punt operaties) 9
2.4. Grafiek maken met plot 10
2.5. Nuttige array-functies 10
2.6. Automatische arrays maken met ones en zeros 11
2.7. Random functies en histogram 11
2.8. Rijen, kolommen en transponeren (.') 12

M3 Programmeren 13

3.1. De for-loop 13
3.2. De if...else..end-constructie 14
3.3. Groter dan, kleiner dan, enz. 14
3.4. While-loop 15
3.5. Switch-case constructie 16
3.6. M-file van het type functie 16

M4 Grafische functies 19

4.1. Grafiek opmaken 19
4.2. Meer grafieken in één plot 19
4.3. Foutengebiedjes plotten 21
4.4. Lijnen trekken in een plot 21
4.5. Meetpunten fitten met Matlab 22
4.6. Kleinste kwadraten fit met fmlsq 23
4.7. Grafisch User Interface (GUI) 24

M5 Grafisch User Interface (GUI) 25

5.1 Properties 25
5.2 Bijbehorende M-file 25
5.3 Handles 26
5.4 Gebruik van de besturingselementen 26

M0 Inleiding

MatLab is een krachtig rekenprogramma, dat zeer veel gebruikt wordt binnen de technische wetenschappen. Het is een veelzijdig programma met veel mogelijkheden:

- een uitgebreide rekenmachine;
- snelle verwerking van grote hoeveelheden data;
- een programmeertaal;
- het maken van een grafisch user interface (GUI);
- een breed scala aan functies en toolboxen voor data-analyse.

En deze opsomming doet bij lange na nog geen recht aan de veelzijdigheid van MatLab.

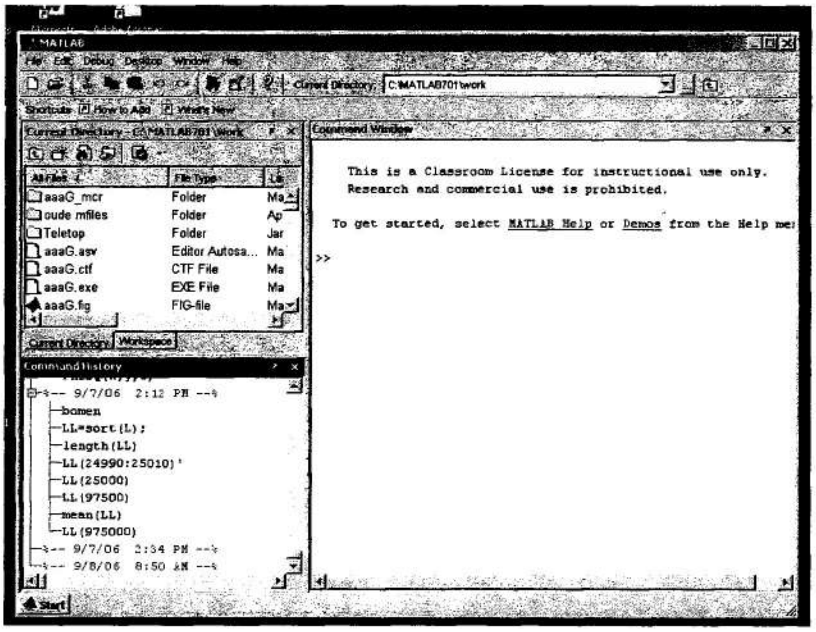
Over deze cursus

Deze cursus is een inleiding in het gebruik van het programma MatLab. Omdat de cursus een klein overzicht van relevante bewerkingen en commando's bevat, kan hij gevolgd worden zonder verder referentiemateriaal te raadplegen. Aangezien deze cursus een onderdeel van het eerstejaarspracticum is, komen met name die onderwerpen aan bod die relevant zijn voor dit practicum. Voorbeelden zijn het uitvoeren van berekeningen op en het grafisch weergeven van meetreeksen, het vinden van *best fits* (met de kleinste kwadratenmethode) en het hanteren van meetfouten.

De cursus Matlab wordt tegelijk aangeboden met Foutenleer. Het is de bedoeling om de voorbeelden in de tekst van het Matlab-gedeelte zoveel mogelijk na te doen in Matlab om een beetje vertrouwd te raken met het programma. Het oefenmateriaal voor Matlab wordt geleverd door het instructiemateriaal van Foutenleer. In de studiewijzer staan aanwijzingen over welke paragrafen en opdrachten in de werkcolleges aan de orde komen.

Opstarten Matlab

Het programma toont drie vensters. Het belangrijkste is de "Command Window". Na de prompt >> kunnen hierin opdrachten getikt worden.



Matlab bewaart de vorige instelling, dus het kan gebeuren dat het openingsscherm er anders uit ziet. Door in het menu "Desktop" bij "Layout" te kiezen voor "Default" kan dit hersteld worden.

Erg belangrijk is de "Current Directory". Daar plaatst en haalt Matlab zijn bestanden.

M1 Eenvoudige Berekeningen

1.1 Directe berekeningen uitvoeren

Net als met een rekenmachine kun je in MatLab berekeningen met getallen uitvoeren. Na de prompt kan de opdracht worden ingetikt. En met **ENTER** wordt hij uitgevoerd. Decimalen worden met een punt en machten van tien met de letter **E** of **e** ingevoerd. Het antwoord krijgt de naam **ans** en kan in een volgende berekening worden gebruikt.

```
>> 64 + 36
ans =
         100
>> ans ^ 3 - 9.99E5
ans =
         1000
```

Onderstaande tabel geeft een overzicht van de rekenkundige bewerkingen. Speciale aandacht verdient het zogenaamde omgekeerde delen.

Overzicht rekenkundige bewerkingen			
optellen	+	23+5	
aftrekken	-	16-12	
vermenigvuldigen	*	3.1*0.85E2	
delen (gewoon)	/	45/9	(= 5)
delen (omgekeerd)	\	25\100	(= 4)
machtsverheffen	^	5^2	

1.2 Rekenen met variabelen

Je wilt de dichtheid bepalen van een legering invar die gemaakt wordt door 64 g ijzer (dichtheid 7,9 g/cm^3) te mengen met 36 g nikkel (dichtheid 8,8 g/cm^3). We berekenen de totale massa en het totale volume van de legering. We voeren de berekening stapsgewijze uit met zelfgekozen variabelen.

```
>> totale_massa = 64 + 36
totale_massa =
         100
>> totaal_volume = 64/7.9 + 36/8.8
totaal_volume =
         12.1922
>> dichtheid = totale_massa / totaal_volume
dichtheid =
         8.2020
```

Zelfgekozen variabelen moeten met een letter beginnen en zijn maximaal 31 tekens (letters of cijfers) lang. Ze zijn hoofdletter-gevoelig en mogen ***geen spaties*** bevatten, wel de underscore.

1.3 Toelichtend commentaar (%)

Als je gebruik maakt van korte namen voor variabelen, is het niet altijd duidelijk wat er ook alweer mee bedoeld wordt. Vaak zet je er dan een korte toelichting bij. Daarvoor gebruik je het procent-teken. Alles wat na het % teken staat, doet niet mee aan berekeningen.

```
>> tm = 64 + 36                     % tm = totale massa
tm =
        100
```

1.4 Echo onderdrukken (;)

Matlab geeft meteen antwoord nadat een commando is ingetikt. Vaak is dat ook de bedoeling, je wilt het resultaat van een berekening zien. Maar soms is het storend, bijvoorbeeld als je eerst variabelen een waarde geeft om er daarna mee te rekenen.
Een puntkomma ; aan het eind van de regel zorgt ervoor dat Matlab de echo onderdrukt.

```
>> a = 7.5                    % zonder puntkomma echoot Matlab
a =
        7.5000
>> b = 2.3;                   % met puntkomma echoot Matlab niet
>> c = a + b;
>> c                          % je wilt toch graag de uitkomst
c =
        9.8000
```

1.5 Wiskundige functies

MatLab kent een groot aantal standaard wiskundige functies, zoals sinus, cosinus en logaritme. De notatie is meestal zoals je zou verwachten. Een uitzondering is de logaritme. Bij berekeningen moet het argument tussen haakjes staan.

Enkele wiskundige functies en constanten

abs(x)	absolute waarde
sin(x), cos(x), tan(x)	sinus, cosinus, tangens
asin(x), acos(x), atan(x)	inverse sinus, cosinus, tangens
round(x)	afronden dichtstbijzijnde getal
exp(x)	natuurlijke exponent: e^x
log(x)	natuurlijke logaritme: ln x
log10(x)	logaritme met grondtal 10
sign(x)	teken van x: 1, 0, -1
sqrt(x)	wortel uit x
pi	het getal π (3,14159...)

Het argument bij goniometrische functies zoals sin(x) dient in radialen te staan:

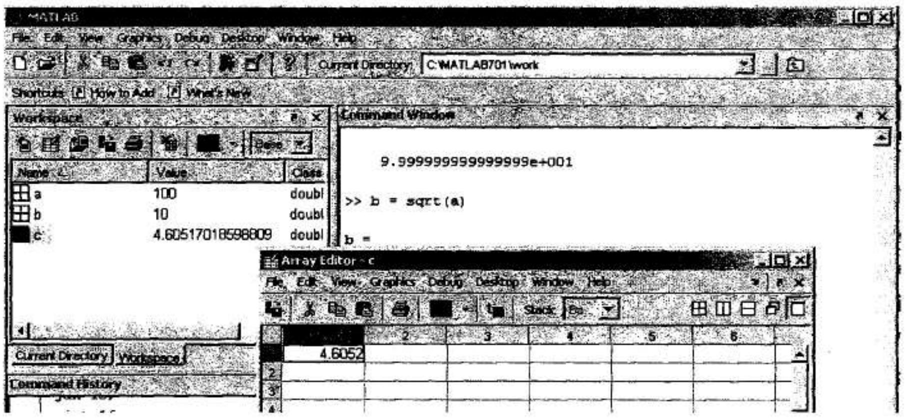
```
>> a = 200 * sin(30*pi/180)                    % argument in radialen
a =
        100.0000
>> b = sqrt(a)
b =
        10
>> c = log(a)                                  % natuurlijke logaritme!
c =
        4.6052
```

1.6 Matlab Workspace

Matlab slaat de zelfgekozen variabelen op in de zogenaamde Workspace. Met het commando **who** krijg je een overzicht van de variabelen.

```
>> who
Your variables are
a              c                  totale_massa
ans            dichtheid
b              totaal_volume
```

Het commando **whos** geeft een gedetailleerd overzicht van de variabelen. Met **clear** wordt de Workspace gewist.
De Workspace window wordt zichtbaar gemaakt door in de window linksboven op de tab "Workspace" te klikken. Met een Array Editor kunnen de variabelen nader bekeken en gemanipuleerd worden. De editor wordt geopend door op een van de blokjes te klikken .



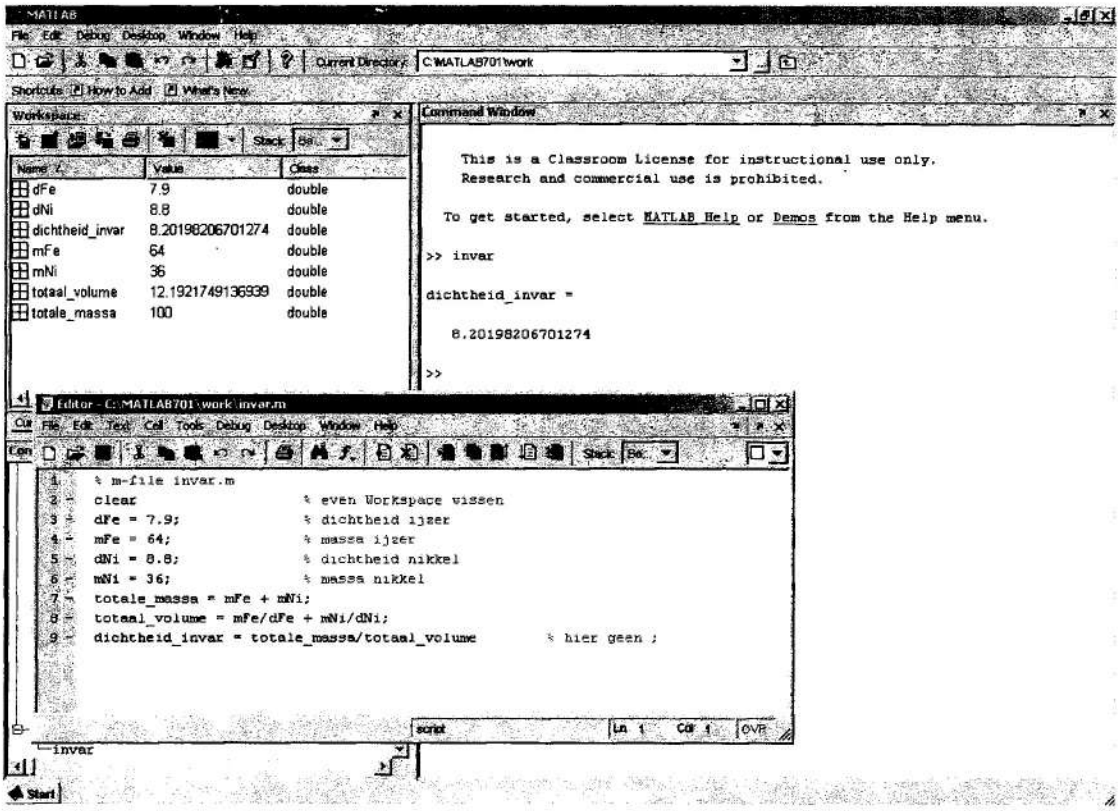

```
Overzicht workspace commando's
who                  geeft opsomming van de gebruikte variabelen
whos                 geeft gedetailleerde informatie variabelen
clear                wist alle variabelen
clear x y            wist alleen de variabelen x en y
```

1.7 M-files

Eerder gedane berekening kun je in de Command Window terugroepen met de pijltjestoets omhoog ↑. Dit is vaak lastig, zeker als je meer commando's wilt herhalen of een keer vergeten bent een puntkomma achter een regel te zetten. De Command-Window loopt snel vol en voorgaande stappen verdwijnen snel uit beeld, zodat het overzicht al gauw zoek is. Daarom maak je meestal even een M-file om berekeningen uit te voeren.
Klik op File - New - M-file en de editor opent zich. Je voert een aantal commando-opdrachten in, de puntkomma aan het eind van de regel is zeer aan te bevelen. Bijvoorbeeld:

```
% m-file invar.m
clear                                       % Workspace wissen
dFe = 7.9;                                  % dichtheid ijzer
mFe = 64;                                   % massa ijzer
dNi = 8.8;                                  % dichtheid nikkel
mNi = 36;                                   % massa nikkel
totale_massa = mFe + mNi;
totaal_volume = mFe/dFe + mNi/dNi;
dichtheid = totale_massa / totaal_volume            %hier geen ;
```

Bewaar de M-file als invar.m en sluit de editor. Op de prompt in de Command Window typ je **invar** en de M-file wordt gerund. Er komt antwoord omdat de laatste regel geen ; heeft.
Het is nu eenvoudig om iets aan de gegevens te veranderen en de berekening opnieuw uit te voeren. Merk op dat de variabelen bij dit type M-file in de Workspace terechtkomen.



M2 Werken met arrays

Een array in MatLab is een rij getallen, bijvoorbeeld meetwaarden of een reeks getallen waarmee je een grafiek wilt tekenen. Matlab beschikt over handige array-functies.

2.1 Arrays invoeren

Er zijn in MatLab een aantal manieren waarop je een array kunt maken. De meest eenvoudige, maar wel tijdrovendste, is het gewoon intypen van elk element met index afzonderlijk. De index begint bij 1 (dus niet bij 0). Er mogen meerdere commando's op één regel staan, mits gescheiden door een komma (geeft echo) of een puntkomma (geen echo).

```
>> t(1) = 5; t(2) = 7; t(3) = 9
t =
        5           7           9
```

Een rij data kan ook in een keer worden ingevoerd. Er moeten rechte haakjes gebruikt worden en de getallen zijn gescheiden door spaties of komma's.

```
>> x = [1   2   2.5   3   4]
x =
        1.0000      2.0000      2.5000      3.0000      4.0000
```

Als de getallen steeds op dezelfde afstand van elkaar liggen zijn er makkelijkere manieren, vooral als je veel getallen moet invoeren. Met de dubbele punt `:` en zonder haakjes geven we de eerste en de laatste waarde met daartussenin de stapgrootte.

```
>> y = 1 : 0.5 : 3                        % stapgrootte is 0.5
y =
        1.0000      1.5000      2.0000      2.5000      3.0000
```

Een andere mogelijkheid om een rij met getallen die steeds op dezelfde afstand liggen te maken is de functie **`linspace`**. Er staan drie getallen tussen haakjes, gescheiden door een komma. De eerste waarde is het beginpunt, de tweede het eindpunt en het derde getal het aantal punten. Let op: het aantal punten is 1 groter dan het aantal tussenstukjes.

```
>> z = linspace(0,10,6)                   % aantal punten = 6
z =
        0       2       4       6       8       10
```

Je kunt een nieuwe array maken met elementen uit andere arrays die in de Workspace staan.

```
>>nieuwe_array = [t(1) x(2:3) y(5) z(6)]
nieuwe_array =
        5.0000      2.0000      2.5000      3.0000      10.0000
```

2.2 Rekenen met arrays (eenvoudig)

"Onze" arrays bestaan meestal uit één serie getallen, bijvoorbeeld een meetreeks. De afzonderlijke getallen van de meetreeks kunnen we bewerken: vermenigvuldigen met een bepaald getal, de sinus ervan nemen, er iets bij optellen, enz. Deze bewerkingen kunnen in één keer op alle data van de array worden toegepast. Het resultaat is een nieuwe array die evenveel elementen heeft.

```
>> clear                                % alle variabelen wissen
x = [0 30 60 90 120 150 180];           % met ; geen echo
>> y = 5 * sin(pi*x/180) + 2
y =
   2.0000   4.5000   6.3301   7.0000   6.3301   4.5000   2.0000
```

Twee even lange arrays kunnen bij elkaar worden opgeteld, van elkaar worden afgetrokken, eventueel met rekenkundige bewerkingen. Het resultaat is een nieuwe array met de bewerking toegepast op de afzonderlijke elementen. Als de arrays niet even lang zijn, lukt dat niet.

```
>> v = linspace(0,10,6); w = v + 10; z = sqrt(w) - v
z =
    3.1623    1.4641   -0.2583   -2.0000   -3.7574   -5.5279
```

2.3 Rekenen met arrays (punt operaties)

Twee dataseries vermenigvuldigen gaat niet op dezelfde gemakkelijke manier als optellen. In het volgende voorbeeld definiëren we een array x en willen vervolgens de waarden van x kwadrateren. Als we dat doen met het gebruikelijke symbool ^ geeft Matlab een foutmelding.

```
>> clear
>> x = linspace(0,10,6);
>> y = x^2                              % gebruik van gewoon dakje ^
??? Error using ==> ^
Matrix must be square.
>> y= x .^ 2                            % gebruik van punt-dakje .^
y =
     0     4    16    36    64   100
```

De afzonderlijke waarden van een meetserie in een array kunnen gekwadrateerd worden met toevoeging van een punt voor de operator. Het wordt dus punt-dakje **.^** in plaats van alleen maar dakje. Hetzelfde geldt voor het onderling vermenigvuldigen en delen van waarden uit verschillende arrays. Vermenigvuldigen wordt punt-ster **.*** en punt-slash **./** staat voor delen.

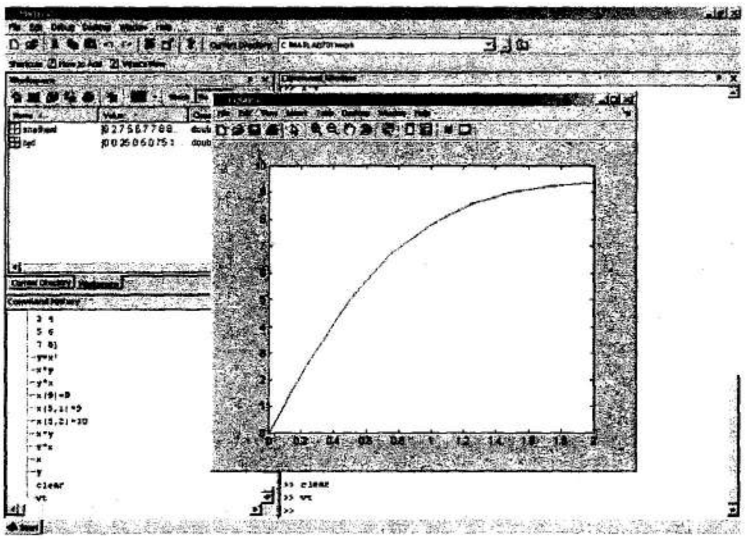
```
>> z = x .* y
z =
      0       8      64     216     512    1000
```

De reden dat gewoon vermenigvuldigen mis gaat ligt in het feit dat Matlab de array beschouwt als een matrix, dat wil zeggen niet één rij getallen maar een aantal rijen. Een vermenigvuldiging wordt dan uitgevoerd als matrix-vermenigvuldiging en dat lukt niet als er maar één rij getallen is. Het gewoon vermenigvuldigen van twee variabelen die geen matrix zijn maar een enkelvoudig getal, lukt wel omdat Matlab deze als een 1 x 1 matrix beschouwt.

2.4 Grafiek maken met plot

Met het commando **plot(x,y)** maakt Matlab een grafiek. De variabelen **x** en **y** zijn even lange arrays. Matlab past de assen van de grafiek aan, maar die kunnen veranderd worden.
In onderstaand voorbeeld voeren we meetgegevens van een tijd-snelheidsmeting in en plotten de v-t grafiek. We gebruiken een M-file om later eventueel veranderingen simpel te kunnen aanbrengen. Bewaar de M-file als **vt.m** en typ na de prompt in de Command Window **vt** in.

```
% M-file vt.m
tijd = 0 : 0.25 : 2;
snelheid = [0 2.7 5.0 6.7 7.8 8.55 9.0 9.24 9.38];
plot(tijd,snelheid)
```



2.5 Nuttige array-functies

Met de functies **max(x)** en **min(x)** kun je naar minimale en maximale waarden zoeken binnen de array x. Ook de statistiek van een meetserie kun je eenvoudig bepalen.

```
Enkele array functies
max(x)        de grootste waarde in een array x
min(x)        de kleinste waarde in een array x
mean(x)       het gemiddelde van de waarden in een array x
std(x)        standaard afwijking van de waarden in een array x
length(x)     het aantal elementen in een array x
sum(x)        de som van alle elementen van array x
sort(x)       de elementen van x worden oplopend gesorteerd
```

Sommige functies kunnen twee outputs te geven. Je moet dan de functie aanroepen met een variabelen-array met twee elementen.

```
>>[a,b] = max(x);          %a = maximumwaarde; b = index
>>[p,q] = sort(x);         %p = gesorteerde array; q = index array
```

Matlab kent nog veel meer array-functies die nuttig kunnen zijn bij data-analyse en dataverwerking. Hiervoor verwijzen we naar de Help van Matlab.

2.6 Automatische arrays maken met ones en zeros

De functies **ones** en **zeros** zijn nuttig bij het maken van nieuwe arrays gevuld met dezelfde waarden, zoals dit bijvoorbeeld bij het intypen van meetfouten regelmatig voorkomt.

```
>> rij_enen = ones(1,7)
rij_enen =
     1     1     1     1     1     1     1
>> fout_in_U = ones(1,5) * 0.05
fout_in_U =
     0.0500    0.0500    0.0500    0.0500    0.0500
>> rij_nullen = zeros(1,8)
rij_nullen =
     0     0     0     0     0     0     0     0
```

De functie **ones(1,n)** maakt een array die uit één rij met n enen bestaat, en **zeros(1,n)** maakt een array die uit één rij met n nullen bestaat. Als we geen rij getallen willen, maar een kolom, dan moeten de 1 en de n van plaats verwisselen. De array is dan een verticale kolom met getallen.
Let op: het gebruik van kolommen in plaats van rijen dient weloverwogen te gebeuren.

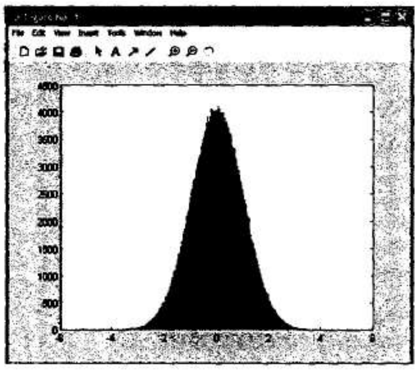
2.7 Random functies en histogram

De functie **rand(1,n)** maakt een array die uit één rij van n getallen bestaat die random (willekeurig) tussen 0 en 1 liggen. Een tweede keer uitvoeren van **rand** geeft weer andere getallen. De functie **randn(1,n)** maakt een array die uit één rij van n getallen bestaat die Gaussisch verdeeld rond het getal 0 liggen, zowel positie als negatief, met een standaard afwijking van 1. Deze worden ook random gegenereerd.

```
% m-file random.m
randn('state',sum(100*clock))        % random generator "schudden"
y = randn(1,1E6);                    % vergeet niet de puntkomma!
hist (y,1000)                        % y in 1000 vakjes indelen
% bewaar de m-file als random.m en typ random in de Command Window
```

In bovenstaand voorbeeld worden 1 miljoen getallen gegenereerd die Gaussisch verdeeld liggen rond 0 met een standaard afwijking van 1. Deze getallen staan in de array y. Met het commando **hist(y,1000)** wordt een histogram van de y-waarden gemaakt. Het getal 1000

geeft het aantal vakjes waarover de y-waarden verdeeld worden. Als we de M-file aanroepen komt het histogram tevoorschijn.
De **randn** functie is erg handig om een serie meetwaarden te simuleren waarbij alleen een toevallige fout een rol speelt.



Als we geen array met random getallen nodig hebben, maar slechts één getal, dan kunnen we de functie **rand** of de functie **randn** aanroepen zonder iets tussen haakje te zetten.

2.8 Rijen, kolommen en transponeren (.')

Matlab geeft soms foutmeldingen bij berekeningen met arrays die onbegrijpelijk lijken. Vaak heeft dat met rijen en kolommen te maken. We illustreren dit met een voorbeeld.
Iemand heeft een meetserie geïmporteerd in Matlab en die is als kolom-array in de Workspace gekomen onder de naam **y**. Vervolgens maakt hij zelf een rij-array **x** met tijdstippen waarop de y-waarden zijn gemeten. De arrays zijn wel even lang.

```
>> y                                    % y is een meetserie (kolom)
y =
      3
      5
      7
      9
     11
x = [0  1  2  3  4];                    % x bevat de tijdstippen (rij)
>> size(x)                              % size geeft [rijen, kolommen]
ans =
      1     5                           % 1 rij en 5 kolommen
>> size(y)
ans =
      5     1                           % 5 rijen en 1 kolom
>> y = y.'                              % rijen en kolommen verwisselen
y =
      3     5     7     9    11         % y is getransponeerd
```

Als je in de Command Window de naam van de array typt, zie je aan de echo of de array een rij of een kolom is. Een andere manier is het opvragen van de dimensies met behulp van het commando **size**. Dit geeft twee output-elementen: het aantal rijen en het aantal kolommen van de array.
Rijen kunnen eenvoudig in kolommen worden omgezet, en omgekeerd, met een punt accentje **.'**. Het verwisselen van rijen en kolommen heet transponeren.

M3 Programmeren

Matlab biedt krachtige rekenfaciliteiten die bij dataverwerking ingezet kunnen worden. Een welkome aanvulling daarop zijn programmeermogelijkheden. De mogelijkheden om in Matlab te programmeren zijn uitgebreid; in deze cursus beperken we ons tot enkele eenvoudige tools. Degenen die vertrouwd zijn met bijvoorbeeld BASIC, zal het bekend voorkomen.

3.1 De for-loop

Met een **for-loop** kunnen we berekeningen of commando's een bepaald aantal keren laten uitvoeren. Er loopt een teller mee die we als index kunnen gebruiken. De syntax is ietsje anders dan bij BASIC: in plaats van "to" komt een dubbele punt **:** en in plaats van "next" komt **end**.
In onderstaand voorbeeld wordt twee keer een **for-loop** gebruikt om alle mogelijke combinaties van lengte en breedte te krijgen.

```
% m-file oppervlak.m
% Gaussisch verdeeld worden 40 lengtes en breedtes gegenereerd en
% bij elk van de 1600 combinaties wordt het oppervlak uitgerekend.
% lengte = 80 met standaard afwijking 0,5
% breedte = 50 met standaard afwijking 0,4

clear
aantal = 40;
lengte = randn(1,aantal)*0.5 + 80;
breedte = randn(1,aantal)*0.4 + 50;
teller = 0;
for k1 = 1:aantal
        for k2 = 1:aantal
               teller = teller + 1;
               opp(teller) = lengte(k1) * breedte(k2);
        end                           % sluiting loop van k2
end                                   % sluiting loop van k1
mean(opp)
std(opp)
hist(opp,50)

% de m-file bewaren als oppervlak.m en tik in de Command Window
% de opdracht oppervlak en deze M-file wordt gerund.
```

De regel `opp(teller) = lengte(k1) * breedte(k2)` rekent elke keer het oppervlak uit; bij de vermenigvuldiging wordt alleen een sterretje gebruikt en geen punt-ster. Dat kan omdat we één element van de array **lengte** vermenigvuldigen met één element van de array **breedte**; het is dus niet een array vermenigvuldiging maar een vermenigvuldiging van twee getallen. Het resultaat is één element van de nieuwe array **opp**.
Door de twee **for-loops** wordt de array **opp** element voor element gevuld. De variabele **teller** fungeert als index voor de array **opp**.

3.2 De if...else...end-constructie

Met een **if.... else...end** constructie kunnen we Matlab berekeningen of commando's laten uitvoeren als aan een bepaalde conditie is voldaan. Ook deze syntax wijkt subtiel af van die bij BASIC: het woordje "then" wordt niet gebruikt en in plaats van "end if" komt alleen **end**. In het onderstaande voorbeeld selecteren we een reeks getallen.

```
% m-file selecteer.m
% we genereren 10000 getallen tussen 0 en 100 en selecteren die
% in de array klein en in de array groot

clear
getal = round(rand(1,10000) * 100);
k = 1; g = 1;
for n = 1:10000
          if getal(n) < 50
                 klein(k) = getal(n);
                 k = k + 1;
          else
                 groot(g) = getal(n);
                 g = g + 1;
          end                                  % sluiting if
end                                            % sluiting loop

disp('          klein          groot')
disp('aantallen')
disp([length(klein),length(groot)])
disp('gemiddeld')
disp([mean(klein),mean(groot)])
disp('standaard afwijking')
disp([std(klein),std(groot)])

% de m-file bewaren als selecteer.m en tik in de Command Window
% de opdracht selecteer en deze M-file wordt gerund.
```

Het commando **clear** in bovenstaande M-file is nodig als we hem vaker laten runnen. De lengte van de arrays **klein** en **groot** zullen namelijk bij elke run iets verschillend zijn. Als een array korter is dan bij de vorige run, zou die array toch zijn vorige lengte houden en zouden enkele oude waarden blijven staan.
Merk op dat de random gegenereerde waarden niet fifty-fifty verdeeld worden over **klein** en **groot**. Er treedt een systematische afwijking op. Probeer hiervoor een verklaring te geven.

3.3 Groter dan, kleiner dan, enz.

De voorwaarde in een **if...else...end** constructie is vaak iets met groter dan of kleiner dan. Hiervoor gebruiken we de zogenaamde relationele operatoren. Matlab heeft een syntax die hier en daar afwijkt van de intuïtie, de is gelijk conditie wordt met twee = tekens geschreven! De uitkomst van een relationele operatie is **true** (= 1) of **false** (= 0).

```
Relationele operatoren in Matlab
 A < B          true als A kleiner is dan B
 A > B          true als A groter is dan B
A <= B          true als A kleiner of gelijk is aan B
A >= B          true als A groter of gelijk is aan B
A == B          true als A gelijk is aan B
A ~= B          true als A ongelijk is aan B
```

3.4 While-loop

Met de **`while-loop`** kunnen we Matlab een serie commando's laten herhalen zolang aan een bepaalde conditie is voldaan. De conditie wordt weergegeven als een relationele operatie (groter dan, kleiner dan, gelijk aan, enz.) of rechtstreeks door een variabele met de logische waarde **`true`** of **`false`**.

```
% m-file raden.m

x = round(rand*8+1);                    % het geheime getal
disp('Raad eens een getal van 1 t/m 9')
keer = 0; raad = 0; foutgeraden = true;
while foutgeraden
   keer = keer+1;
   raad = input(['Aantal keer raden = ',num2str(keer),'; Gokje: ']);
    if raad == x                        % Let op dubbel is-teken!
        foutgeraden = false; disp('GOED!!')
    else
        disp('FOUT!!')
    end
end
disp(['Je krijgt als cijfer een ',num2str(11-keer)])

% de m-file bewaren als raden.m en tik in de Command Window
% de opdracht raden en deze M-file wordt gerund.
```

Het is zeer belangrijk dat de loop niet eeuwig duurt. De variabele waarop gecheckt wordt, moet dus binnen de loop de waarde **`false`** kunnen krijgen. Het is daarom verstandig om in de ontwerpfase een reserve in te bouwen om een eeuwig durende loop te voorkomen. Te denken valt aan een teller: een variabele die bij elke loop eentje opgehoogd wordt en bijvoorbeeld bij 10 de check-conditie **`false`** maakt.
In de regel die begint met **`raad = input ( [`**... wordt de gebruiken gevraagd om input via het toetsenbord te geven. Datgene wat de gebruiker intypt wordt als waarde toegekend aan de variabele **raad**, waarmee het programma verder aan de slag gaat.

3.5 Switch-case constructie

De **switch-case-end** constructie biedt de mogelijkheid om bij verschillende waarden van een variabele een apart commando uit te voeren. Het geeft een heel overzichtelijke programma-code.

```
% m-file beoordeling.m

cijfer = input('Typ hier het laatste gehaalde tentamencijfer: ');

switch cijfer
    case {1,2,3,4,5}
        disp([' Het cijfer ',num2str(cijfer),' is onvoldoende.'])
    case 6
        disp([' Het cijfer ',num2str(cijfer),' is voldoende.'])
    case {7,8}
        disp([' Het cijfer ',num2str(cijfer),' is goed.'])
    case {9,10}
        disp([' Het cijfer ',num2str(cijfer),' is zeer goed.'])
    otherwise
        disp(' ! Cijfers van 1 t/m 10 en gehele getallen.')
end

% de m-file bewaren als beoordeling.m en tik in de Command Window
% de opdracht beoordeling en deze M-file wordt gerund.
```

3.6 M-file van het type functie

Matlab biedt de mogelijkheid om zelf een nieuwe functie in te voeren en die vanuit de Command Window aan te roepen. Dit wordt gedaan in de vorm van een M-file.
In paragraaf 1.7 hebben we kennisgemaakt met het type M-file dat alleen uit een serie commando's bestaat waarvan het handig is om ze in een apart bestandje te hebben. De M-file die we in deze paragraaf bespreken is van het type functie en verschilt op een paar punten van de gewone M-file:

- De eerste regel begint met het woord **function**, gevolgd door een toewijzing.
- Je kunt input-variabelen aan de functie meegeven.
- De functie kan een output hebben.
- De variabelen van de functie zelf komen niet in de Workspace.

Matlab heeft zelf een grote hoeveelheid functies (zoals **sin(x)**, **mean(x)**, **sqrt(x)**, **plot(x,y)**) die niet als naam van de eigen nieuwe functie gekozen kunnen worden.
De functie **uitbijter** in onderstaand voorbeeld controleert of er in een opgegeven datareeks een uitbijter zit. Een uitbijter is een waarde die (erg) veel afwijkt van de andere waarden en beter weggelaten kan worden omdat hij er te veel uitspringt. De functie **uitbijter** krijgt de waarde 0 als er geen uitbijter is en de waarde van de index van de uitbijter als dat wel het geval is.
Als een functie naar behoren werkt, kan die als M-file ter beschikking gesteld worden aan andere gebruikers.

```
% uitbijter.m

function ub = uitbijter(x,n)

switch nargin                     % nargin is aantal input argumenten
    case 0
        error('Geef tussen haakjes een data array op!')
    case 1
        disp('Geen criterium opgegeven. Default = 5.')
        norm=5;
    case 2
        norm=n;
end

aantal_punten = length(x); gemiddelde_x = mean(x);
[grootste,welke_g] = max(x); [kleinste,welke_k] = min(x);
if abs(max(x)-gemiddelde_x) > abs(min(x)-gemiddelde_x)
    verdachtn = welke_g;
else
    verdachtn = welke_k;
end
overig = [x(1:verdachtn-1) x(verdachtn+1:aantal_punten)];
afwijking = (abs(x(verdachtn)-mean(overig)))/std(overig);
if afwijking > norm
    ub = verdachtn;
else
    ub = 0;
end

% de m-file bewaren als uitbijter.m en roep in de Command Window
% de functie aan, bijvoorbeeld welke = uitbijter(mijndata,3).
```

De functie wordt in de Command Window aangeroepen met één of twee input argumenten: een data array en het getal waarmee bepaald wordt of een bepaalde waarde uitbijter is of niet. Als het tweede argument ontbreekt, wordt daarvoor 5 genomen. De functie zoekt eerst de waarde met de grootste afwijking. Dan maakt hij een array `overig` waarin die verdachte ontbreekt. Vervolgens wordt gekeken hoever de verdachte waarde afwijkt van het gemiddelde van `overig`, uitgedrukt in de standaard afwijking van `overig`. Deze afwijking wordt vergeleken met het opgegeven criterium.
De waarde voor het criterium is enigszins arbitrair, maar moet niet te klein zijn, zeker niet bij kleine datasets. Om een verdachte waarde als uitbijter te bestempelen is het beter de datareeks uit te breiden dan het criterium te verlagen.
De functie `sorteer(x,y)` in het volgende voorbeeld sorteert de meetpunten **(x,y)** naar opklimmende grootte in de waarde **x**. Dit is erg handig als er aanvullende meetpunten zijn toegevoegd aan een datareeks.

```
% sorteer.m

function [sx,sy]=sorteer(x,y)

if nargin < 2
    error('Twee arrays nodig: x-waarden en y-waarden!')
end
aantal = length(x);
if length(y) ~= aantal
    error('Arrays ongelijk in lengte')
end

[hulparray,volgorde] = sort(x);
sx=hulparray;                        % toekennen output sx
for n = 1:aantal
    sy(n) = y(volgorde(n));          % toekennen output sy
end

% de m-file bewaren als sorteer.m en roep in de Command Window
% de functie aan met b.v. [svolt,sampere] = sorteer(volt,ampere).
```

Met het commando **`[hulparray,volgorde] = sort(x);`** wordt Matlab gewongen om twee output argumenten te geven. Het eerste, **`hulparray`**, is een array waarin de waarden van de array **x** in opklimmende grootte komen te staan. Het tweede output argument is **`volgorde`**, een array waarin de indexen staan die de gesorteerde waarden hadden in de oorspronkelijke array **x**. Hiermee kunnen we dus de bijbehorende y-waarden zoeken. De gesorteerde arrays worden toegekend aan de dummy output arrays **`sx`** en **`sy`**, dit zijn feitelijk de functiewaarden die **`sorteer`** als ouput teruglevert.
De M-file kan worden aangeroepen in de Command Window. De output van de functie moet op een handige manier opgevangen worden door expliciet om twee outputs te vragen.

```
>> volt = [4.5710 5.5810 7.7170 8.9230 6.5760 10.1070 2.5460 11.0270
1.5300 12.0280 0.6130 3.5220];
>> ampere = [0.2620 0.2940 0.3530 0.3830 0.3220 0.4120 0.1870 0.4330
0.1370 0.4550 0.0810 0.2250];
>> [svolt,sampere] = sorteer(volt,ampere);
>> plot(svolt,sampere)
```

Let op: het apart sorteren van de **x** en **y** array zou niet-bijelkaarhorende puntenparen kunnen opleveren.

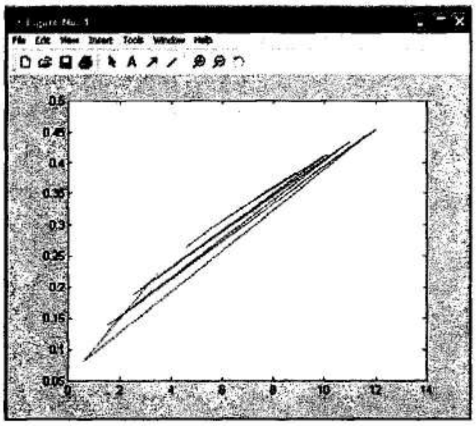
M4 Grafische functies

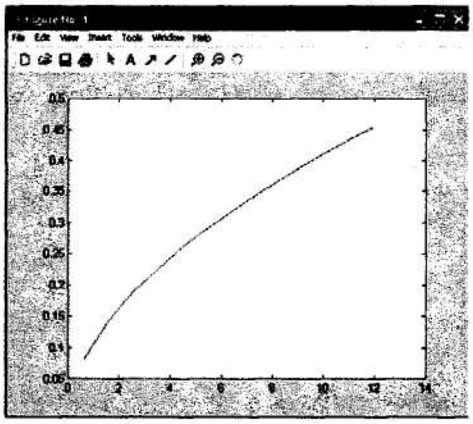
Bij natuurwetenschappelijke toepassingen is het gebruikelijk om gegevens grafisch te presenteren als meetpunten in een assenstelsel. De Matlab-functie die dit op een eenvoudige manier doet is **`plot(x,y)`** waarbij x en y data-arrays van gelijke lengte zijn. De eerstgenoemde array wordt als x-coördinaat beschouwd en de als tweede genoemde array als y-coördinaat.

4.1 Grafiek opmaken

Met het commando **`plot(x,y)`** wordt het assenstelsel getekend waarin de puntenparen (x,y) worden uitgezet. Matlab gebruikt defaultinstellingen voor bijvoorbeeld de kleur, de lengte van de assen en de weergave van de punten. Meestal levert dit al een acceptabel resultaat op. Er zijn enkele veelvoorkomende schoonheidsfoutjes.

- Matlab tekent lijntjes tussen de meetpunten, soms is dat een probleem.
- Matlab laat de X- en Y-as niet standaard doorlopen tot aan de oorsprong.
- Als de geplotte meetpunten niet opklimmend van grootte zijn, vertoont de lijn die Matlab er standaard doorheen legt allerlei rare lussen.





De lussen kunnen verholpen worden door de meetpunten te sorteren op volgorde van grootte of door alleen punten te displayen en geen lijn. Bovenstaande grafieken zijn van een ongesorteerde en een gesorteerde dataset.
De opmaak van grafieken kan aangepast worden via de menu-knoppen in de Window waarin de grafiek verschijnt. Via **`Edit`** kunnen de assen worden aangepast, er kan een raster getoond worden, de puntenparen kunnen als aparte punten weergegeven worden, enz. Met **`Insert`** kunnen labels bij de assen worden aangebracht, tekst in de grafiek toegevoegd worden, handmatig lijnen en pijlen worden getrokken, enz.
Bij nieuwere versies van Matlab kunnen communicatie-windowtjes opgeroepen worden door op de figuur te klikken. Klikken op de meetpunten biedt bijvoorbeeld de mogelijkheid om de getrokken lijn weg te halen en te vervangen door markers voor de meetpunten.

4.2 Meer grafieken in één plot

Met het commando **`figure`** maakt Matlab een nieuwe Figure Window aan voor de volgende plot. Als je dat niet doet, wist Matlab de huidige Figure Window en begint bij het commando **`plot(x,y)`** met een nieuw assenstelsel. Met het commando **`hold`** houdt Matlab de vorige grafiek vast en tekent in dezelfde Figure Window de nieuwe daarbij in. Eventueel volgende grafieken worden er ook bij in getekend. Pas als opnieuw het commando **`hold`** komt, wordt het plotscherm opgegeven. Een alternatief om hetzelfde te doen zijn de commando's **`hold on`** om het plotscherm vast te zetten en **`hold off`** om het weer vrij te geven.

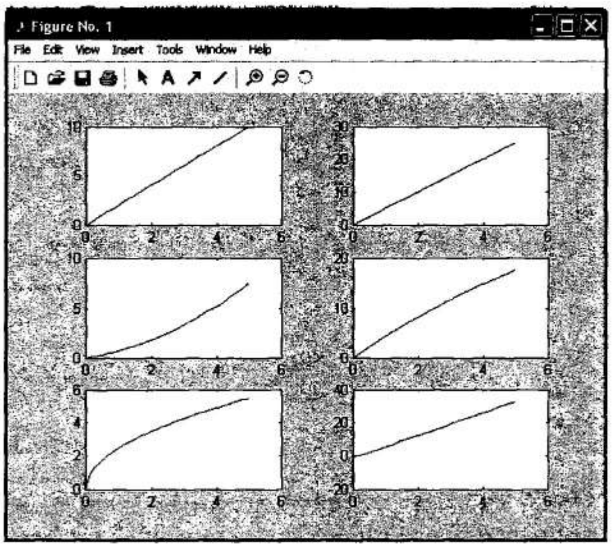
```
>> plot(x1,y1)                  % Eerste grafiek plotten
>> hold                         % Equivalent met hold on
Current plot held
>> plot(x2,y2)                  % Tweede grafiek erbij
>> hold                         % Equivalent met hold off
Current plot released
>>
```

Twee grafieken in één assenstelsel kunnen ook direct gemaakt worden. Het commando **plot(x1,y1,x2,y2)** zal de getallenparen (x1,y1) en (x2,y2) in dezelfde plot tonen.

```
>> plot(x1,y1,x2,y2)
>>
```

Het is ook mogelijk om twee of meer grafieken onder elkaar of naast elkaar te plotten. Dan laten we het commando **plot(x,y)** voorafgaan door het commando **subplot**. Het commando **subplot** krijgt drie getallen mee: het eerste is het aantal grafieken dat onder elkaar komt, het twee is het aantal grafieken dat naast elkaar komt en het derde getal is het volgnummer van de grafiek die geplot wordt.
Intypen van **subplot(3,2,5)** betekent dat de Figure Window opgedeeld is in zes kleine grafiekjes: drie onder elkaar en twee naast elkaar. En dat grafiekje nummer 5 aan de beurt is geplot te worden bij het daaropvolgende **plot** commando.

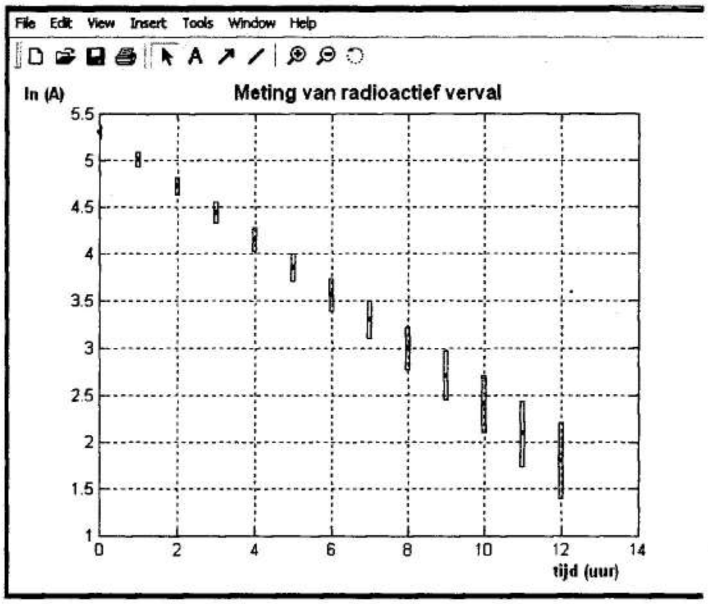
```
>> subplot(3,2,1); plot(t,x1)        % linksboven = 1
>> subplot(3,2,2); plot(t,x2)        % rechtsboven = 2
>> subplot(3,2,3); plot(t,x3)        % enz.
>> subplot(3,2,4); plot(t,x4)
>> subplot(3,2,5); plot(t,x5)
>> subplot(3,2,6); plot(t,x6)        % rechtsonder = 6
>>
```



4.3 Foutengebiedjes plotten

Met de speciaal voor het TNW-practicum ontwikkelde functie **fmerrorbox(x,y,dx,dy)** is het mogelijk om een grafiek te maken van een reeks meetpunten (x,y) met hun foutengebiedje (dx,dy). Het is een m-file van het type functie en maakt gebruik van het Matlab-commando plot, zodat de grafieken op de gebruikelijke manier kunnen worden opgemaakt. De gebruiker moet behalve de arrays x en y ook nog de arrays dx en dy maken. Als x en y geen rechtstreekse meetwaarden zijn, maar bijvoorbeeld gekwadrateerd zijn of er is een logaritme genomen, dient er speciale aandacht te zijn voor het correct berekenen van de foutengebiedjes van de geplotte grootheden.

```
% verval.m
t = 0 : 1 : 12;
dt = zeros(1,13);                 % foutengebiedje tijd = 0
A = [200, 149, 112, 84, 63, 47, 35, 27, 20, 15, 11, 8, 6];
dA = sqrt (A);                    % meetfout activiteit
LnA = log (A);                    % lineariseren van de grafiek
dLnA = dA ./ A;                   % fout in geplotte grootheid op Y-as
fmerrorbox (t,LnA,dt,dLnA)
% deze M-file bewaren als verval.m en in de Command Window
% de opdracht verval intikken.

>>verval
>>
```



4.4 Lijnen trekken in een plot

Eén van de icoontjes in de Figure Window biedt de mogelijkheid om een lijn aan te brengen in de plot. Soms is die mogelijkheid voldoende voor wat de gebruiker wil. Als hij echter een specifieke lijn wil die door bepaalde punten gaat of een bepaalde helling heeft, zal hij nauwkeuriger te werk moeten gaan. Matlab trekt in een bestaande plot een rechte lijn van

(x1,y1) naar (x2,y2) met het commando **line([x1 x2],[y1 y2])**. Tussen haakjes moet dus eerst een array staan met de beide x-waarden, dan een komma en dan een array met de beide y-waarden. Als de arrays meer dan twee elementen hebben trekt Matlab lijnstukken tussen de verschillende punten.

```
>> x = [0 9]; y = [5 8];          % beginpunt (0,5) eindpunt (9,8)
>> line(x,y)                      % Matlab trekt een lijn
```

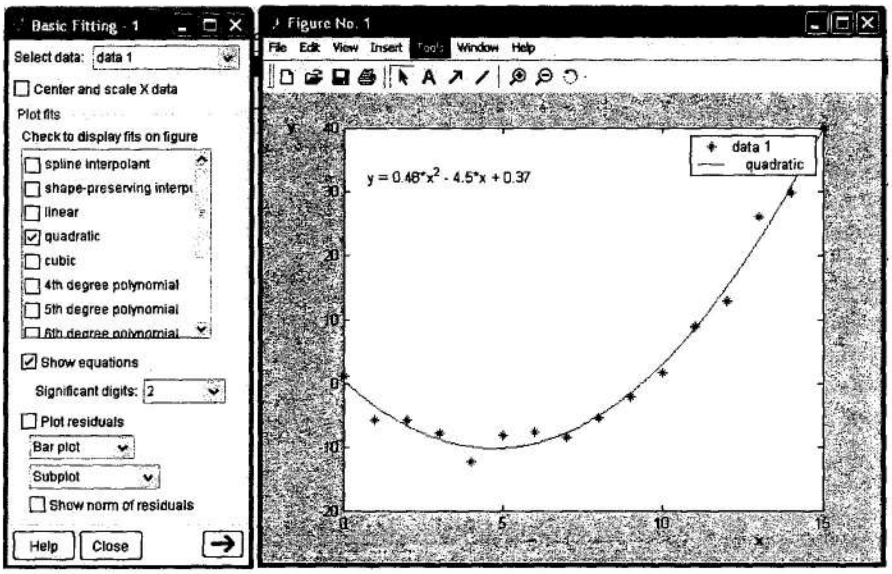
Er kan ook interactief een lijn getrokken worden. Met het commando **ginput(2)** moet de gebruiker twee keer in de plot klikken. De x-waarden en de y-waarden kunnen in een array afgevangen worden om te gebruiken voor het trekken van een lijn.

```
>> [xklik,yklik] = ginput(2);     % klik 2 keer in de plot
>> line(xklik,yklik)              % lijn tussen de geklikte punten
```

4.5 Meetpunten fitten met Matlab

Matlab kan geplotte meetpunten fitten met een eerste-, tweede- of hogere graads functie, een zogenaamde polynoom. In de Figure Window onder de menuknop **Tools** kan **Basic Fitting** aangeklikt worden. Er verschijnt een nieuw windowtje, hierin kan de gebruiker de graad van de polynoom opgeven en eventuele voorkeuren aanklikken.
De gebruiker dient erg kritisch te zijn in het gebruik van polynomen en het beoordelen van de fitresultaten. Het programma zal namelijk altijd met een "best fit" komen, maar dat zegt niets over de betrouwbaarheid en nauwkeurigheid van de fit. Zelfs een perfecte fit kan betekenisloos zijn: door tien willekeurige punten gaat namelijk altijd een negende graads polynoom! Of anders gezegd: door twee meetpunten gaat altijd een rechte lijn.
In het algemeen geldt dat er (theoretische) redenen moeten zijn om te kiezen voor een bepaalde fit.

Zestien (x,y) punten worden gefit met een 2° graads functie.

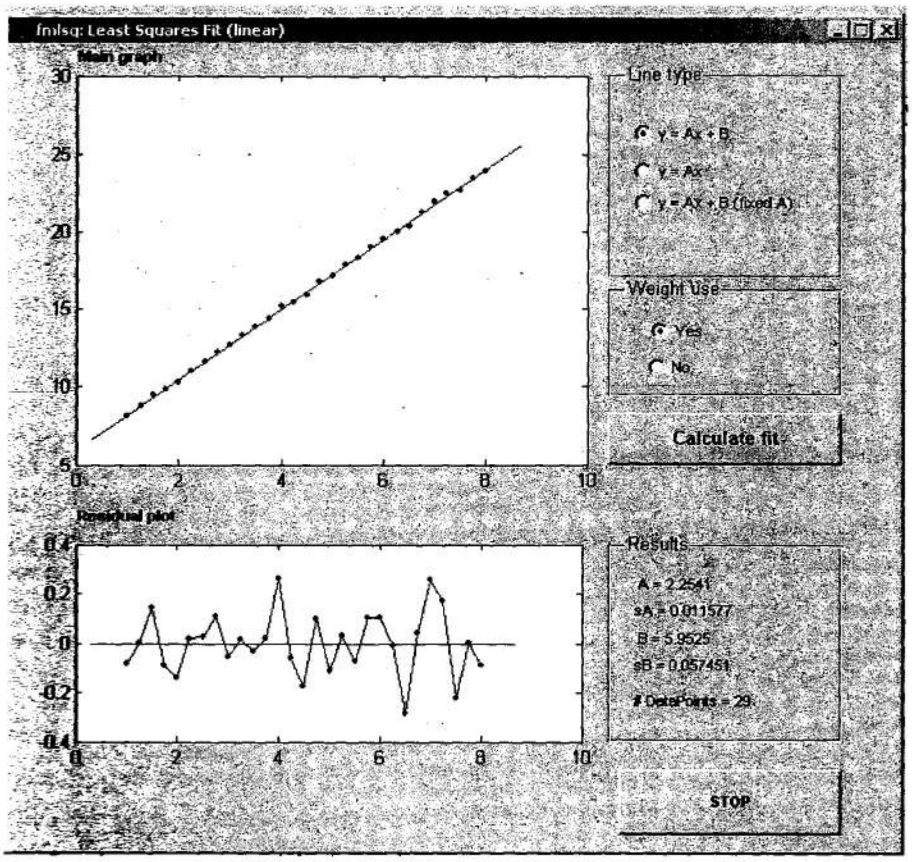


4.6 Kleinste kwadraten fit

Ten behoeve van het TNW-practicum is de functie **`fmlsq`** ontwikkeld om het lineaire verband of het gelineariseerde verband tussen twee grootheden te onderzoeken. Behalve de best passende rechte lijn geeft de functie ook de nauwkeurigheid van de fit. De functie kent verschillende uitvoeringen, afhankelijk van het gebruiksdoel.
De basisfunctie **`fmlsq`** toont een grafiek van de meetpunten in een user interface. Hij wordt aangeroepen met het commando **`fmlsq(x,y,w)`**. Hierin zijn x en y de arrays met x- en y-waarden van de puntenparen. Het derde argument, de array w, bevat de onderlinge gewichten van de meetpunten. De array w moet even lang zijn als x en y. De arrays x en y zijn noodzakelijk als input-argument; de array w niet. Als w niet wordt meegegeven krijgen alle meetpunten het gewicht 1.
De gebruiker kan kiezen met welk type lijn de fit wordt uitgevoerd:

- y = Ax + B
- y = Ax
- y = Ax + B met gegeven helling A.

Verder kan de gebruiker aangeven of er wel of geen gewichten gebruikt moeten worden.

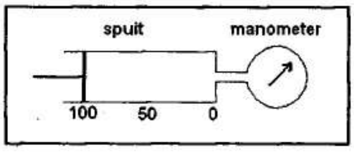


Behalve de meetpunten en de parameters van de best passende rechte lijn wordt ook een grafiek van de residuen getoond: de afwijking tussen de gemeten en de berekende y-waarden. De basisfunctie **`fmlsq`** geeft geen output. Uitprinten kan via een screendump.
Bij geavanceerd gebruik van de kleinste kwadraten fit heeft de gebruiker wel output nodig. Hij kan dan gebruik maken van een van de aanvullende functies **`fmlsqAB`**, **`fmlsqA`** of **`fmlsqB`**. Deze functies genereren geen grafisch user interface; de gebruiker krijgt ook geen grafiek te zien. Deze functies worden als volgt aangeroepen.

- **`BesteLijn = fmlsqAB(x,y,w)`** waarbij x en y de data-arrays zijn (wel nodig) en w de array met gewichten (mag weggelaten worden).

- **BesteLijn = fmlsqA(x,y,w)** waarbij x en y de data-arrays zijn (wel nodig) en w de array met gewichten (mag weggelaten worden). De lijn gaat door de oorsprong.
- **BesteLijn = fmlsqB(x,y,helling,w)** waarbij x en y de data-arrays zijn (wel nodig), helling een getal (wel nodig) en w de array met gewichten (mag weggelaten worden). De beste lijn heeft de opgegeven helling.

De output bestaat in elk van de drie gevallen uit de rij-array BesteLijn met vier elementen: de helling van de best passende lijn, de standaardafwijking in de helling, de asafsnede en de standaardafwijking in de asafsnede.
De array BesteLijn kan gebruikt worden voor verdere berekeningen of als output in de Command Window.
In onderstaand voorbeeld bevat de M-file **gasdruk.m** meetwaarden van de gasdruk bij verschillende volumes van een spuit. De druk wordt afgelezen op een manometer met een onbekend eigen volume. De druk zou volgens de algemene gaswet omgekeerd evenredig moeten zijn met het totale gasvolume. We verwachten een rechte lijn als we 1/P uitzetten tegen V. De best passende lijn wordt gevonden met **fmlsqAB**. De M-file wordt gerund door in de Command Window **gasdruk** te typen.

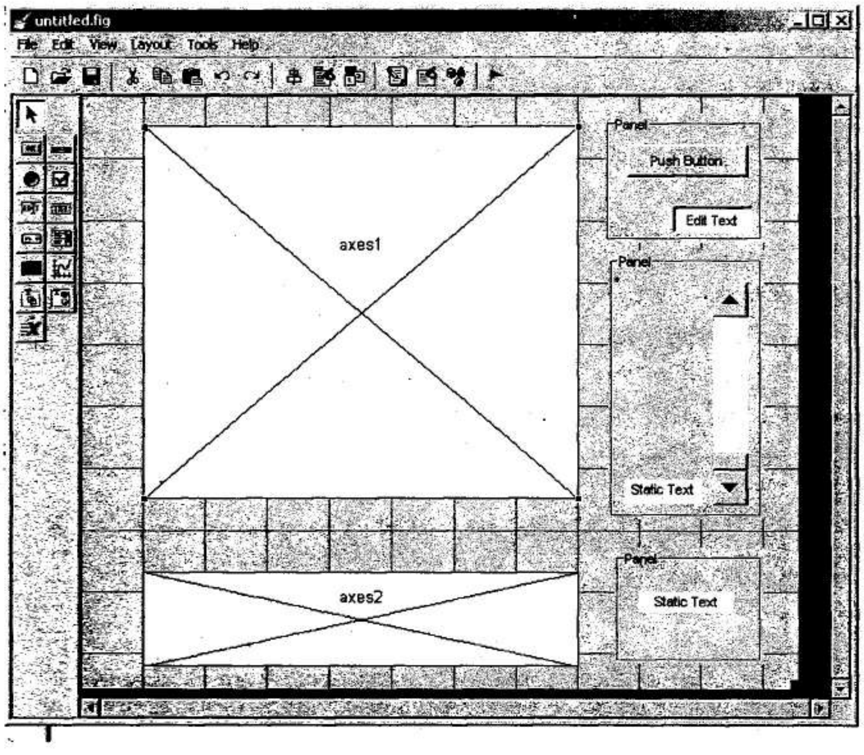



```
% gasdruk.m

V = 100 : -10 : 0;              % Volume spuit in mL
P = [1.10 1.14 1.37 1.46 1.66 1.95 2.30 2.85 3.72 5.45 9.95]; %bar
Xas = V;                        % volume komt op de X-as
Yas = 1./P;                     % lineariseren van de grafiek
w = P.^2;                       % array met gewichten
BesteLijn = fmlsqAB(Xas,Yas,w)            % kleinste kwadraten fit

% M-file bewaren als gasdruk.m en in Command Window gasdruk intypen

>> gasdruk
BesteLijn =

     0.0082652     8.2234e-005     0.10096     0. 0018847
>>
```

De best passende lijn gaat niet door (0,0) omdat horizontaal het volume van de <u>spuit</u> is uitgezet en niet het totale gasvolume. Uit de helling volgt de temperatuur of het aantal mol gas. Het onbekende volume van de manometer en de verbindingsbuis kan met de outputgegevens berekend worden.

M5 Grafisch User Interface (GUI)

Met Matlab kan vrij eenvoudig een grafisch user interface (GUI) gemaakt worden.
Kies File -> New -> GUI -> Create new GUI -> Blank GUI en er komt een werkvel tevoorschijn waarop besturingselementen geplaatst kunnen worden. Zie de schermafdruk.



Veel gebruikte besturingselementen zijn:
Axes: een assenstelsel waarin een plot gemaakt kan worden.
Push Button: een OK knop waarop de gebruiker met de muis kan klikken.
Edit Text: een textvakje waar de gebruiken iets in kan typen.
Slider: een schuif.
Static Text: een textvakje waar het programma iets in kan zetten.
Panel: een framepje waarbinnen een aantal elementen gegroepeerd kunnen worden.

5.1 Properties

Van elk besturingselement kunnen de eigenschappen handmatig ingesteld worden via de Property inspector, op te roepen via de rechter muiskop. Het gaat om naam, kleur, grootte, positie, lettertype, enz. Men plaatst eerst een besturingselement op het werkvel, ongeveer op de gewenste plaat met ongeveer de gewenste grootte. En later verfijnt men de positie en afmetingen met de Property inspector. Dit verfijnen gebeurt meestal als een van de laatste stappen. Eerst schrijft men een bijbehorende M-file en test die. Het komt vaak voor dat men gaandeweg een nieuw idee krijgt, waardoor het verfijnen en het opmaken van de interface pas aan het eind komt. Elk besturingselement behoudt zijn formele naam waarmee het door de programmeur aangeroepen kan worden.

5.2 Bijbehorende M-file

Als de besturingselementen geplaatst zijn, moet de GUI gesaved worden. Hij wordt opgeslagen onder een te kiezen naam met de extensie .fig. Tegelijk met het opslaan genereert Matlab een M-file met dezelfde naam met de extensie .m waar de gebruiker zijn programma

in kwijt kan. De M-file bevat al het skelet van een programma gebaseerd op de geplaatste besturingselementen.
Als het programma klaar is, wordt de M-file gesaved en kan vervolgens gerund worden. De gebruiker krijgt dan de GUI op zijn scherm en kan daarmee interactief dingen doen.

5.3 Handles

Matlab heeft zijn eigenaardigheden waar een beginnend gebruiker even aan moet wennen. Een GUI heeft afzonderlijke elementen en de M-file moet die onderdelen op verschillende plaatsen in het programma kunnen uitlezen en besturen. Ook zijn er data-arrays en grootheden die overal in het programma beschikbaar moeten zijn. Deze informatie wordt doorgegeven via zogenaamde **handles**. De formele namen van besturingselementen beginnen met dit woord: **handles.edit1, handles.slider1, handles.axes1, handles.text1.**
De programmeur moet ook via deze weg zijn data beschikbaar maken, de meetwaarden heten dan bijvoorbeeld **handles.xwaarde** en **handles.ywaarde**.
Berekeningen worden vaak in een subroutine uitgevoerd. Men zet aan het eind programma een functie die aangeroepen kan worden. De **handles** worden als argument meegegeven.

```
% Aanroepen van de functie die de berekeningen doet
Berekeningen(handles)

% Definieren van de functie die de berekeningen uitvoert
function Berekeningen(handles)
(programma)
```

5.4 Gebruik van de besturingselementen

Besturingselementen die zich daarvoor lenen, kunnen met **get** worden uitgelezen en met **set** van een waarde worden voorzien. Alles wat in textvakjes staat wordt als tekst beschouwd, ook als het een getal is. Tekst is een serie symbolen en heet **'String'**. Er is een functie die een string inhoudelijk leest en interpreteert als een getal: **str2double**. Het besturingselement slider geeft wel een getal.

```
% uitlezen van Edit Text element en er een getal van maken
waarde1 = str2double(get(handles.edit1,'String'));

% een getal displayen in een Static Text element
waarde2 = 123.876;
set(handles.text1,'String',waarde2);

% een schuifje uitlezen en displayen in een Static Text element
waarde3 = get(handles.slider1,'Value');
set(handles.text2,'String',waarde3);

% een grafiek plotten begint met het kiezen van het assenstelsel
axes(handles.axes1)
plot(handles.meetx,handles.meety,'.','Markersize',6)
```